CMPUT 328 Fall 2024 Assignment 4

Object Detection and Semantic Segmentation

Worth 12% of the total weight

Overview

In this assignment, you are going to do object detection and semantic segmentation on the MNIST Double Digits RGB (MNISTDD-RGB) dataset. This dataset contains RGB images of size 64×64 where each has two digits from 0 to 9 placed randomly inside it as shown in Figure 1. Your task is to figure out which digits are contained in each image and where are they located along with their precise pixel-wise segmentation masks as in Figure 2.

Dataset

MNISTDD-RGB is a synthetic dataset generated by training a GAN on MNIST dataset and using the trained model to generate new MNIST-like 28 x 28 grayscale images of digits which are then colorized with randomly chosen colors. Next, two of these colorized images are placed at random locations inside a 64 x 64 image with a randomly chosen background color. Finally, one of 3 types of noise – Gaussian, uniform, impulse - is randomly added to some images while others are left without noise such that roughly a quarter of the images are noiseless while a quarter each have one of the three types of noise.

The dataset is divided into 3 subsets - train, validation and test - containing 55K, 5K and 10K samples respectively. A sample consists of:

- Image: A $64 \times 64 \; x \; 3$ image that has been vectorized to a 12288-dimensional vector (Figure 1)
- Labels: A vector containing two numbers in the range [0, 9] which are the two digits in the image.
  - These two numbers are always in ascending order. For example, if digits 7 and 5 are in an image, then this two-vector will be [5, 7] and not [7, 5]
- Bounding boxes: A 2×4 matrix which contains two bounding boxes that mark the locations of the two digits in the image.
  - The first row contains the location of the first digit in labels and the second row contains the location of the second one.
  - Each row of the matrix has 4 numbers which represent $[x\_min,\ y\_min,\ x\_max,\ y\_max]$ in this exact order, where:
    - x_min = column of the top left corner
    - y_min = row of the top left corner
    - x_max = column of the bottom right corner
    - y_max = row of the bottom right corner
- Semantic Segmentation Mask: A 64×64 image with pixel values in the range [0, 10], where 0-9 represent the corresponding classes while 10 represents the background (Figure 2)
- Instance Segmentation Mask: A 64×64 image with pixel values in the range [0, 2], where 1 and 2 represent the first and second digits respectively while 0 represents the background (Figure 3)

Each set comprises a single *npz* file which can be read using *numpy.load* to obtain the corresponding arrays stored as *numpy.ndarray* instances.

Following are detailed descriptions of the 5 arrays, where N is the number of samples:

- *images*: 2D array with dimension $[N, 12288]$ and containing the vectorized images. Each row is a single vectorized RGB image of size 64 x 64 x 3
- *labels:* 2D array with dimension $[N, 2]$ and containing the labels. Note that the labels are always in ascending order in each row.
- *bboxes:* 3D array with dimension $[N, 2, 4]$ and containing the bounding boxes.
- *semantic_masks:* 2D array with dimension [N, 4096] and containing the pixel-wise class labels. Background pixels are 10.
- *instance_masks:* 2D array with dimension [N, 4096] and containing the pixel-wise object IDs. The object whose label comes first (and whose bounding box is in the first row) has ID 1 while the other one has ID 2. Background pixels are 0.

Each row in both masks is a single vectorized image of size 64 x 64. The digit on top in the source image is also on top in the mask so the other one is occluded by it, i.e., any pixel occupied by both digits will have the label of the top one.

Task

You are provided with the train and validation sets that can be downloaded from e-class as 2 files: *train.npz* and *valid.npz*.

The test set is not released to better represent real-world conditions where test data is not available before the trained model is deployed.

You are also provided three python files: *A4_main.py*, *A4_utils.py* and *A4_submission.py*. The latter has a single function ***detect_and_segment*** that you need to complete. It takes a single matrix containing all test (or validation) images as input and returns three *numpy* arrays - ***pred_class***, ***pred_bboxes***, ***pred_seg*** - containing the classification labels, detection bounding boxes and segmentation masks respectively in the same format as described in the dataset section above.

To reiterate, the two digits in each row of *pred_class* must be in ascending order and the two rows in *pred_bboxes* corresponding to that image must match this ordering too.

Similarly, the digit that is on top in the source image should also be on top in *pred_seg* so the other one is occluded by it, i.e., any pixel occupied by both digits will have the label of the top one.

A4_main.py can be run to evaluate the performance of your method in terms of:

- classification accuracy: percent of digits classified correctly
- detection accuracy: intersection over union (IOU) of the detected boxes with the ground truth boxes
- segmentation accuracy: percent of foreground pixels classified correctly

It also contains code to visualize MNISTDD-RGB images with both ground truth and predicted bounding boxes, labels and masks drawn on them which might help you figure out how to convert the *npz* files into a format suitable for use with existing models.

A4_utils.py contains helper code for visualization.

You are free to add any other functions or classes you need, including any additional files imported from *A4_submission.py*. Just make sure to submit all files needed to run your code including any trained checkpoints.

You can use any machine learning or image processing method to solve this problem. One of the main objectives of this assignment is for you to learn how to adapt an existing model to solve a new problem so there is no restriction or guideline as to what algorithms or libraries you can or should use. You are also not restricted to using *pytorch* for this assignment.

You have the option to either train two separate models for object detection and semantic segmentation or a single instance segmentation model that does both.

Following are some practical constraints related to submission and evaluation:

- Your model must be able to process at least 10 images / sec to avoid run time penalty as explained in the next section
- E-class has an upload limit of 400 MB so your model must be small enough to fit in this size.
- Your model must be able to run on Colab GPU
  - Available GPU memory on Colab can vary from session to session so make sure to run your model several times before submitting to ensure that it is not so close to the memory limit that it fails to run during evaluation.
  - Memory needed for training is often significantly more than testing and it is only the latter that matters for evaluation.
- The marking TA will **not** be installing any packages during evaluation so if your method requires libraries that are not installed in Colab by default, you must install them in your code. Any time needed for such installation will count towards your run time requirement.

There are no part-marks in this assignment, e.g. for submitting training code or checkpoints with non-functioning ***detect_and_segment***, so any submission not satisfying the above criteria or any of the marking criteria below will simply get **no marks**.

Marking

Your marks will be determined by the following four criteria:

- Classification Accuracy
  - Linear scaling from **60% to 95%**
- Detection IOU
  - Linear scaling from **50% to 85%**
- Segmentation Accuracy
  - Linear scaling from **50% to 80%**
  - It will be computed only over **foreground pixels**, i.e., pixels marked as background in both the ground truth and predicted masks will be ignored.

For each of the above criteria, your marks will scale linearly between two thresholds *A* and *B***.** Submissions with accuracy, IOU, or segmentation accuracy <= *A* and >= *B* will respectively get no marks and full marks for that criterion.

- Time Penalty:
  - No penalty for **time <= 1000 secs** (or **speed >= 10 images / sec**)
  - No marks for **time > 10,000 secs** (or **speed < 1 image / sec**)
  - Linear scaling of penalty for **1000 secs < time <= 10,000 secs,** that is,
    penalty = (time - 1000) / (10,000 -1000)

In order as to give equal weightage to the detection and segmentation tasks, your overall marks will be computed as:

$$score = ((classification + IOU) / 2 + segmentation) / 2$$

The final marks will be computed by applying the time penalty to the overall score:

$$marks = (1 - penalty) \times score$$

What to Submit

You need to submit a single zip file containing the modified *A4_submission.py* along with any other files or checkpoints needed to run it on the test set. Testing will be done by running *A4_main.py* and it is your responsibility to ensure that your code can be imported and run without errors.

Please do not include training code in your submission. If you do and your model starts training during evaluation, it will get no marks for this assignment.

Due Date

The due date is **Sunday, November 10 at 11:59 pm**. The assignment is to be submitted online on e-class.

Collaboration Policy

This must be your own work. Do not share or look at the code of other people (whether they are inside or outside the class). Do not copy code from the Internet though you can look for ideas to write your own code. You can talk to others that are in the class about solution ideas but not in so much detail that you are verbally sharing or hearing about or seeing the code. You must cite whom you talked to or what websites you consulted to write your code in the comments of your programs.

This policy also applies to the submitted checkpoints (if any) so make sure to train your own model even if you use a standardized architecture (e.g., in *torchvision* or *torch.hub*) that is shared with other submissions. We will be able to determine from the contents of the submitted checkpoints if they have been plagiarized.

The usage of ChatGPT is allowed, but not recommended. Additionally, we reserve the right to evaluate any student's submission further through a viva if we have reason to believe that they do not understand the solution that they have submitted.

Submission Checklist

All submissions will be used in an automated evaluation system and any failure to run, for any reason, will be heavily penalized and might lead to the submission getting a zero.

- Make sure that your code runs without errors on Colab, whether the GPU is enabled or disabled.
- Remove any notebook specific lines (e.g., mounting Google drive) or any other code that can produce an error when it is imported.
- Do not submit *A4_main.py, A4_utils.py* or any of the *npz* files.
- Do not rename *A4_submission.py* to anything else (including changing any part of it to upper or lowercase).
- Training and validation files will not be available during testing so make sure to remove any code that loads these.
- Set paths for loading weights (if any) relative to *A4_submission.py* in your submitted zip file.
- Make sure that the testing batch size is small enough to not run out of memory on Colab.
- All included files and folders must be in the root of the zip file (instead of being contained inside another folder).
- Submit a zip file instead of any other archive format (e.g., rar, tar or tar.gz)
- E-class has a maximum upload size limit of **400 MB** so your submission must be small enough to fit in this size.
    - Google Drive (or any cloud storage) links for any part of your submission will not be accepted.

Resources

Following are links to some popular Pytorch-based object detection and semantic segmentation repositories on GitHub, though there is no guarantee that any of these models will run on Colab or provide sufficient performance. It is your responsibility, if you choose to use any of these, to adapt them to work within the constraints of this assignment.

Many of these are also included within torchvision and pytorch hub.

- Detectron2
- Open MMLab Detection Toolbox
- YOLO v5 / v8
- Swin Transformer Object Detection
- DETR
- pytorch-retinanet

- Deeplab v3
- UNet
- Open MMLab Semantic Segmentation Toolbox
- Swin Transformer Semantic Segmentation

Paper / code collections:

- Awesome Object Detection
- Object Detection with Deep Learning
- Awesome Semantic Segmentation
- SemanticSegmentation_DL

Tutorials:

- SSD: Single Shot MultiBox Detector | A PyTorch Tutorial to Object Detection
- Torchvision Object Detection Finetuning Tutorial
- PyTorch object detection with pre-trained networks
- Training an object detector from scratch in PyTorch
- Train your own object detector with Faster-RCNN & PyTorch

❖ Custom Object Detection using PyTorch Faster RCNN

Figure 1: Visualization of the 25 images generated by the same GAN used for generating the images in MNISTDD-RGB. The two digits in an image may partially or completely overlap each other, though complete overlap only happens in a small fraction of images. The digits may also be of the same class.

Bounding boxes for both digits are shown in black.

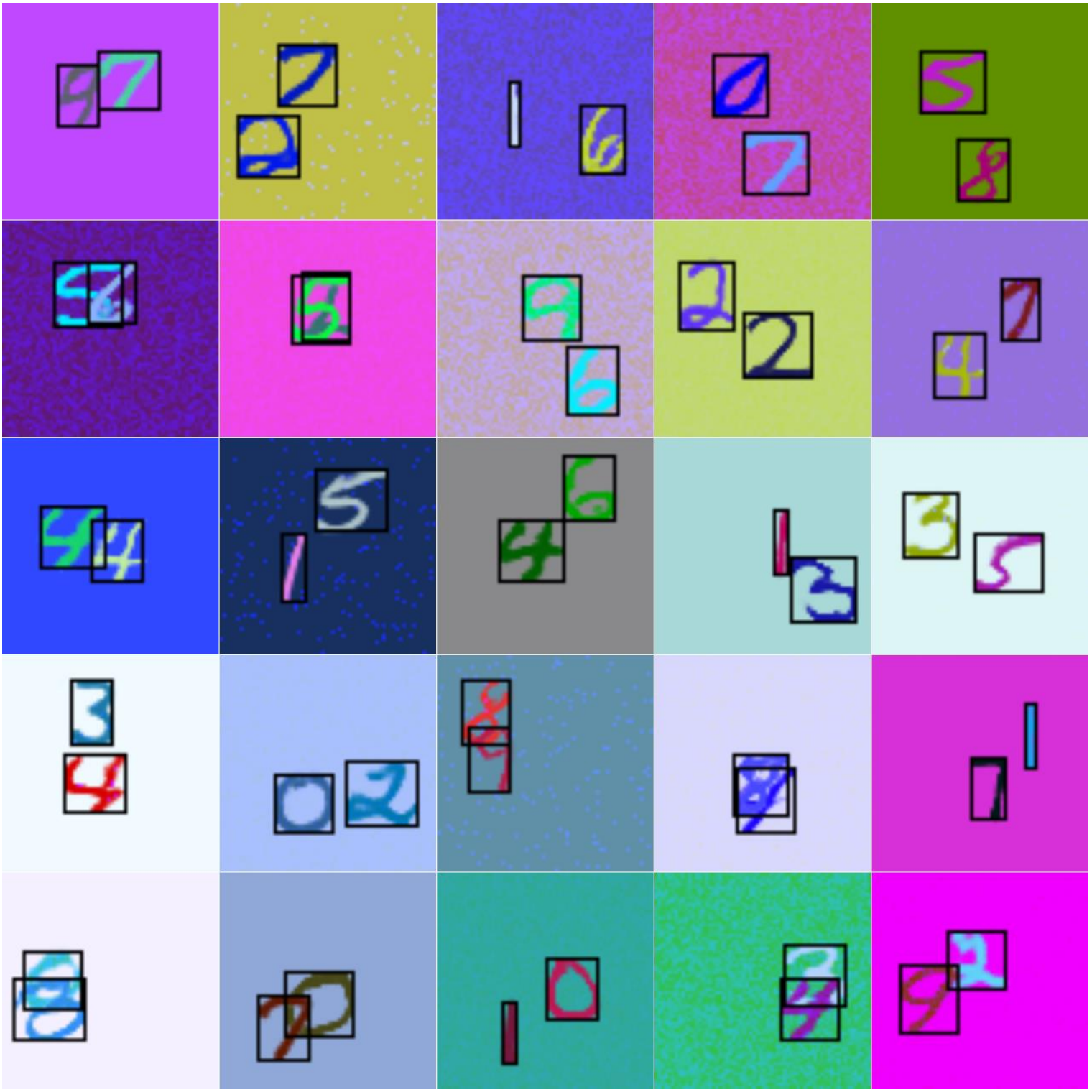

Figure 2: Visualization of semantic segmentation masks for the images from Figure 1.

Following colors are used here to show the various digits:

0: red 1: green 2: blue 3: magenta 4: cyan 5: yellow 6: purple 7: forest_green 8: orange 9: maroon

Note that these colors are used only for visualization here and the actual value of each pixel in the mask is the same as the digit it corresponds to. Background pixels have a value of 10 and are shown here in black.

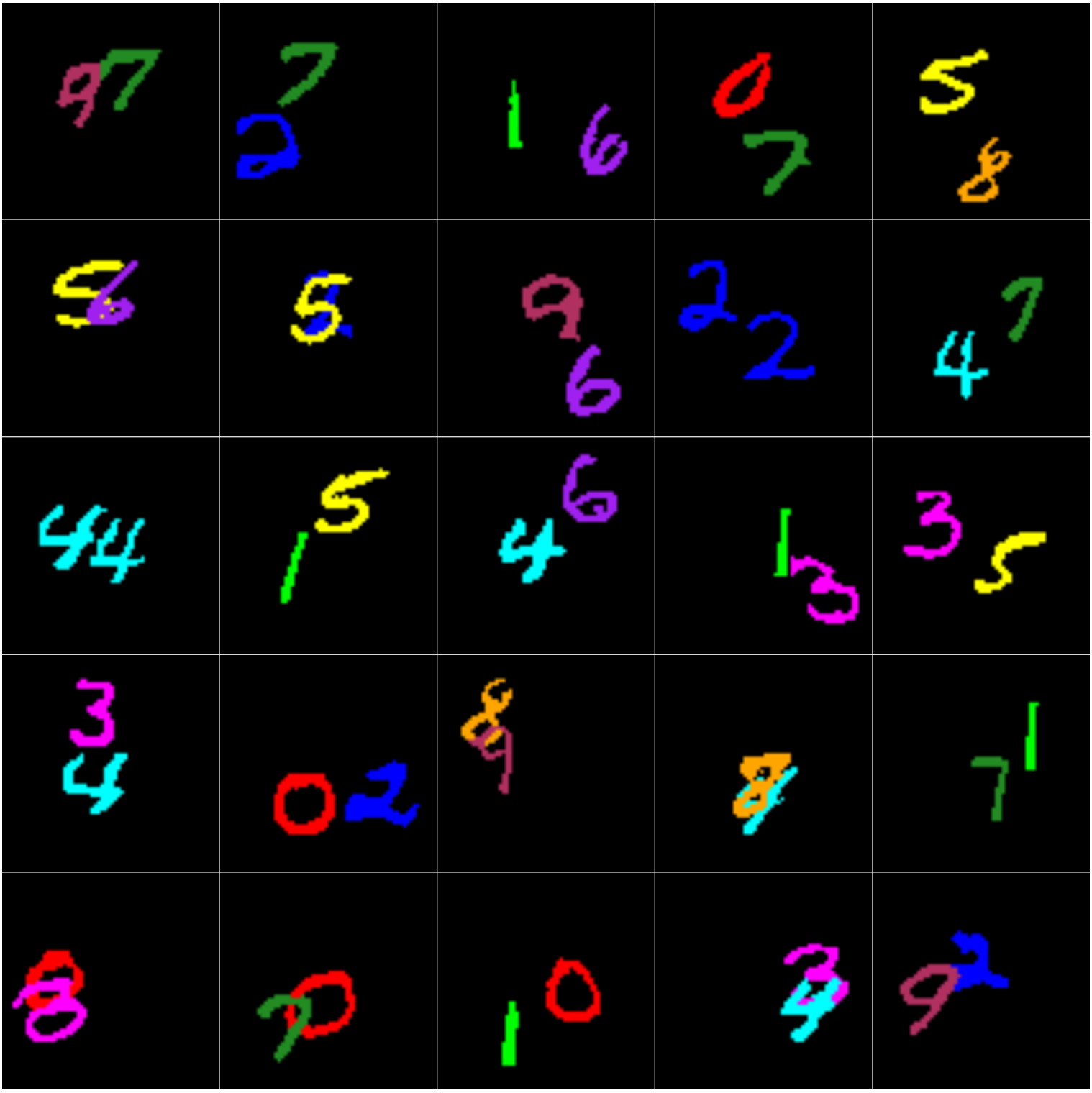

Figure 3: Visualization of instance segmentation masks for the images from Figure 1. Background is shown in black while red and green respectively show the first and second digits respectively.

Note that these colors are used only for visualization here and the actual value of each pixel in the mask is 0 for background, 1 for the first digit and 2 for the second one.